

AUT02_09 Identity en Discos Entrega Final

Margot Hernández González
2º DAW B
Curso 2024-2025

Índice

Informe	3
Seeding	3
Acceso por roles	5
Asignar rol cuando se registra un nuevo usuario	6
Mostrar vista solo a determinados roles	6
Creamos el controlador Admin a mano	7
Conclusiones	9
Enlaces útiles	9

Informe

Seeding

- Roles

```
0 referencias
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    //Seeding users
    List<IdentityRole> roles = new List<IdentityRole>
    {
        new IdentityRole
        {
            Name = "Administrator",
            NormalizedName = "ADMINISTRATOR"
        },
        new IdentityRole
        {
            Name = "User",
            NormalizedName = "USER"
        },
        new IdentityRole
        {
            Name = "Platinum",
            NormalizedName = "PLATINUM"
        }
    };
    modelBuilder.Entity<IdentityRole>().HasData(roles);
}
```

- Users

```
List<IdentityUser> users = new List<IdentityUser>
{
    new IdentityUser
    {
        UserName = "admin@chinook.com",
        NormalizedUserName = "ADMIN@CHINOOK.COM",
        Email = "admin@chinook.com",
        NormalizedEmail = "ADMIN@CHINOOK.COM",
        EmailConfirmed = true
    },
    new IdentityUser
    {
        UserName = "user@chinook.com",
        NormalizedUserName = "USER@CHINOOK.COM",
        Email = "user@chinook.com",
        NormalizedEmail = "USER@CHINOOK.COM",
        EmailConfirmed = true
    },
    new IdentityUser
    {
        UserName = "platinum@chinook.com",
        NormalizedUserName = "PLATINUM@CHINOOK.COM",
        Email = "platinum@chinook.com",
        NormalizedEmail = "PLATINUM@CHINOOK.COM",
        EmailConfirmed = true
    }
};
modelBuilder.Entity<IdentityUser>().HasData(users);

var passwordHasher = new PasswordHasher<IdentityUser>();
users[0].PasswordHash = passwordHasher.HashPassword(users[0], "Inma24.");
users[1].PasswordHash = passwordHasher.HashPassword(users[1], "Inma24.");
users[2].PasswordHash = passwordHasher.HashPassword(users[2], "Inma24.");
```

El passwordHasher lo hacemos después y tenemos que hacerlo uniendo al user de la lista Users con la contraseña.

- Usuarios y roles

```
modelBuilder.Entity<IdentityUserRole<string>>().HasData(
    new IdentityUserRole<string> { UserId = users[0].Id, RoleId = roles[0].Id },
    new IdentityUserRole<string> { UserId = users[1].Id, RoleId = roles[1].Id },
    new IdentityUserRole<string> { UserId = users[2].Id, RoleId = roles[2].Id }
);

base.OnModelCreating(modelBuilder);
}
```

Unimos el user[0] (que es admin) con roles[0] (que es el administrator). Y así con user y platinum. Terminar con base.OnModelCreating.

- Luego hacemos un Add-Migration M0002
- Update-Database

Acceso por roles

- Es importante añadir este builder en program.cs

```
builder.Services.AddDbContext<Margot_ChinookContext>();

builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddRoles<IdentityRole>() //Soporte para roles
    .AddEntityFrameworkStores<Margot_ChinookContext>();
```

- En el controlador que queremos restringir por roles

```
namespace hernandezmargot_nasica.controllers
{
    [Authorize(Roles = "Administrator, Platinum")]
    public class AdminMargotController : Controller
    {
        private readonly Margot_ChinookContext _context;
        private readonly UserManager<IdentityUser> _userManager;

        0 referencias
        public AdminMargotController(Margot_ChinookContext context, UserManager<IdentityUser> userManager)
        {
            _context = context;
            _userManager = userManager;
        }

        0 referencias
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Asignar rol cuando se registra un nuevo usuario

- En register.cshtml.cs

```
0 referencias
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl ??= Url.Content("~/");
    ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
    if (ModelState.IsValid)
    {
        var user = CreateUser();

        await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
        await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
        var result = await _userManager.CreateAsync(user, Input.Password);

        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");

            var userId = await _userManager.GetUserIdAsync(user);

            //añadimos esta línea
            await _userManager.AddToRoleAsync(user, "User");

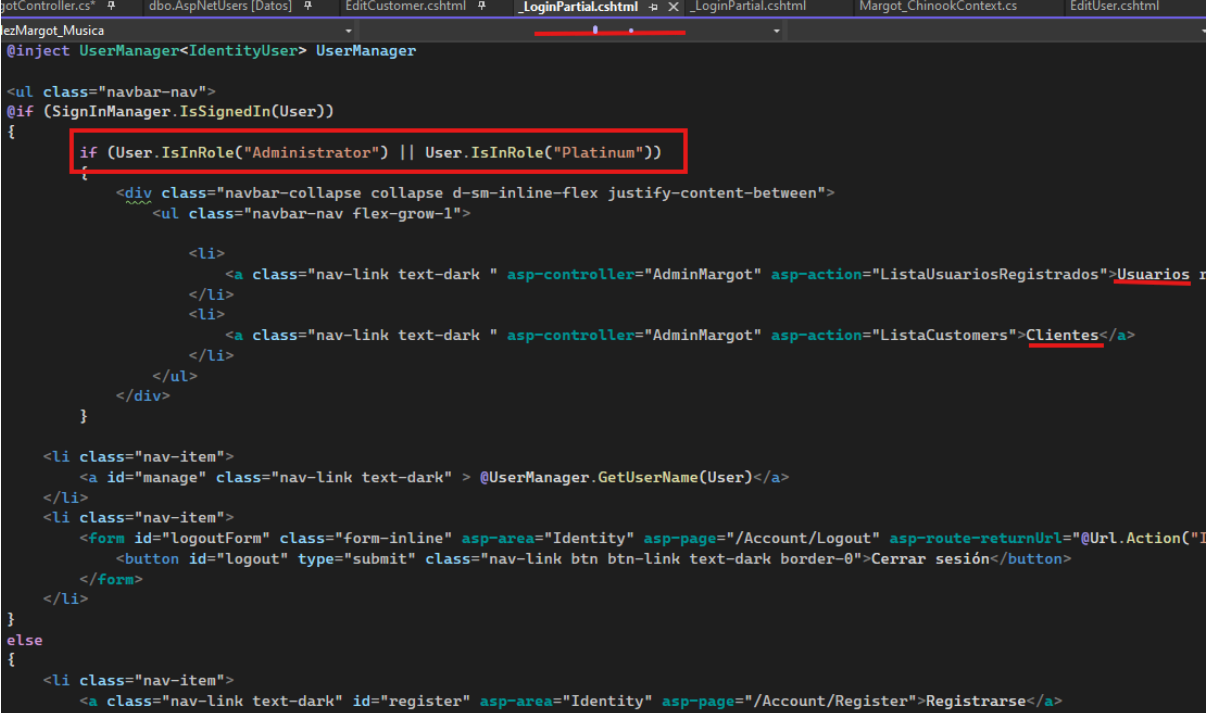
            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
            var callbackUrl = Url.Page(
                "/Account/ConfirmEmail",
                pageHandler: null,
                values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
                protocol: Request.Scheme);
        }
    }
}
```

Mostrar vista solo a determinados roles

- primero ponemos en el _layout

```
</div>
@Html.Partial("_LoginPartial")
</div>
</nav>
```

- En el _loginPartial configuramos la vista. En el ejemplo consigo que solo los usuarios con roles Platinum y Admin puedan acceder a clientes y usuarios registrados



```

@inject UserManager<IdentityUser> UserManager

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    if (User.IsInRole("Administrator") || User.IsInRole("Platinum"))
    {
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
            <ul class="navbar-nav flex-grow-1">
                <li>
                    <a class="nav-link text-dark" asp-controller="AdminMargot" asp-action="ListaUsuariosRegistrados">Usuarios</a>
                </li>
                <li>
                    <a class="nav-link text-dark" asp-controller="AdminMargot" asp-action="ListaCustomers">Clientes</a>
                </li>
            </ul>
        </div>
    }

    <li class="nav-item">
        <a id="manage" class="nav-link text-dark" > @UserManager.GetUserName(User)</a>
    </li>
    <li class="nav-item">
        <form id="logoutForm" class="form-inline" asp-area="Identity" asp-page="/Account/Logout" asp-route-returnUrl="@Url.Action("Logout", "Account", Context.Request.Headers.ToQueryString())">
            <button id="logout" type="submit" class="nav-link btn btn-link text-dark border-0">Cerrar sesión</button>
        </form>
    </li>
}
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" id="register" asp-area="Identity" asp-page="/Account/Register">Registrarse</a>
    </li>
}
}

```

Creamos el controlador Admin a mano

- Traemos el modelo de la tabla users, tenemos que especificar el contexto que ya tenemos creado y eliminar en -ContextDir Data porque ya tenemos uno (si lo dejamos lo sobrescribe). Generé otra carpeta para el modelo Users
 - Scaffold-DbContext "Server=(localdb)\MSSQLLocalDB; Database=Chinook; Trusted_Connection=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models2 -Tables.AspNetUsers -Context **Margot_ChinookContext** -DataAnnotations
- Dentro crearemos las acciones correspondientes con la autorización de roles

- Para borrar los usuarios tuve que añadir en el Context, dentro del OnModelCreating

```
//-----  
  
modelBuilder.Entity<IdentityUserRole<string>>()  
    .HasOne<IdentityUser>()  
    .WithMany()  
    .HasForeignKey(userRole => userRole.UserId)  
    .onDelete(DeleteBehavior.Cascade);
```

Con esto borramos las relaciones entre usuarios y roles (tabla NetUserRoles)

- Luego editamos el _LoginPartial

```
@using Microsoft.AspNetCore.Identity  
  
@inject SignInManager<IdentityUser> SignInManager  
@inject UserManager<IdentityUser> UserManager  
  
<ul class="navbar-nav">  
@if (SignInManager.IsSignedIn(User))  
{  
    @if (User.IsInRole("Administrator") || User.IsInRole("Platinum"))  
    {  
        <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">  
            <ul class="navbar-nav flex-grow-1">  
                <li>  
                    <a class="nav-link text-dark" asp-controller="AdminMargot" asp-action="Index">Inicio</a>  
                </li>  
                <li>  
                    <a class="nav-link text-dark" asp-controller="AdminMargot" asp-action="Index">Inicio</a>  
                </li>  
            </ul>  
        </div>  
    }  
  
    <li class="nav-item">  
        <a id="manage" class="nav-link text-dark" asp-area="Identity" asp-page="/Account/Manage/Index">Gestionar</a>  
    </li>  
    <li class="nav-item">  
        <form id="logoutForm" class="form-inline" asp-area="Identity" asp-page="/Account/Logout/LogoutForm">  
            <button id="logout" type="submit" class="nav-link btn btn-link text-dark border-0">Cerrar sesión</button>  
        </form>  
    </li>  
}  
else
```

La vista se va a modificar según estes logueado o no. Y luego según tu rol se mostrará las funcionalidades del AdminController

Conclusiones

Trabajar con Identity en este proyecto ha sido un desafío significativo, principalmente debido a la base sobre la que estaba construido. El trabajo de música inicial contenía varios errores que complicaron el desarrollo y que, al postergar su corrección, dificultaron aún más el avance.

Implementar Identity y sus funcionalidades (como la gestión de usuarios, roles y seguridad) me obligó a investigar profundamente para comprender su funcionamiento y resolver problemas que surgieron en el camino. Aunque fue un proceso complejo, me permitió aprender mucho sobre la integración de sistemas de autenticación en proyectos existentes y la importancia de contar con una base sólida desde el principio.

Además de trabajar en las funcionalidades de autenticación, gestión de usuarios y roles, he aprovechado esta oportunidad para integrar GitHub como herramienta de control de versiones.

Enlaces útiles

- <https://learn.microsoft.com/es-es/aspnet/core/security/authentication/identity?view=aspnetcore-8.0&tabs=visual-studio>
- <https://learn.microsoft.com/es-es/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-8.0&tabs=visual-studio#scaffold-identity-into-a-razor-project-with-authorization>
- <https://www.c-sharpcorner.com/article/seed-data-in-net-core-identity/>
- <https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-8.0>
- <https://iberasync.es/roles-en-asp-net-core-e-identity/>
-